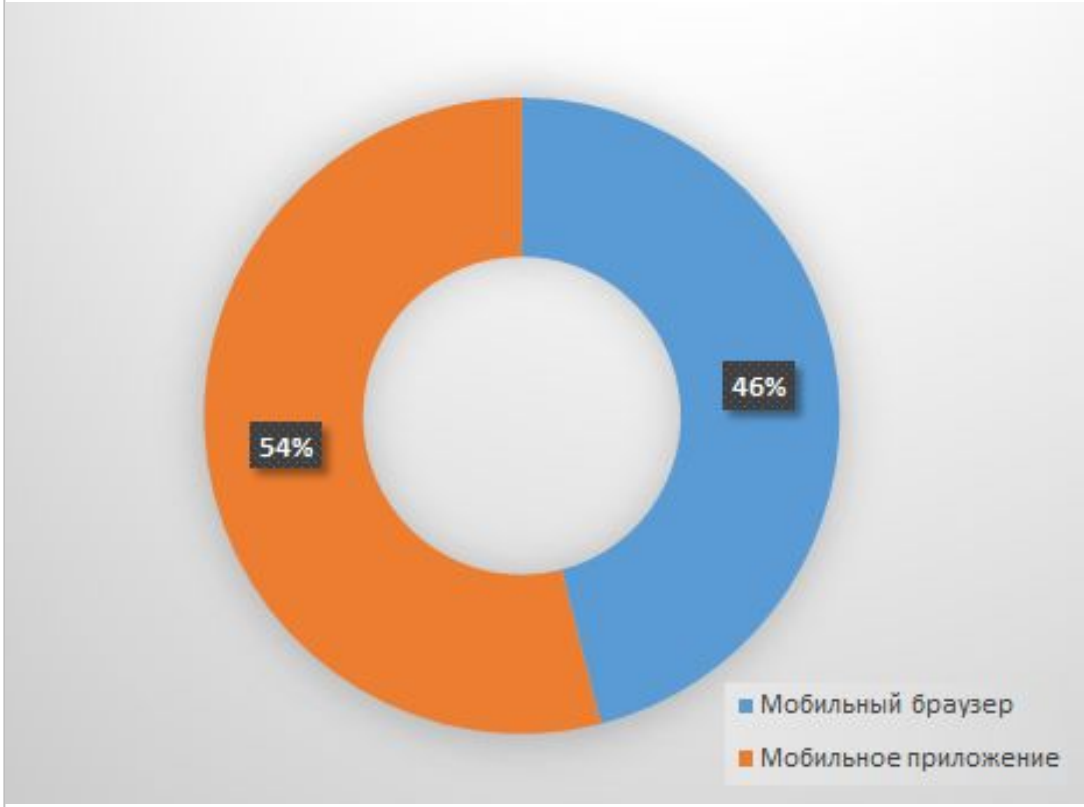
Всё = сервисы?

Голованов Константин / Руководитель разработки платформы

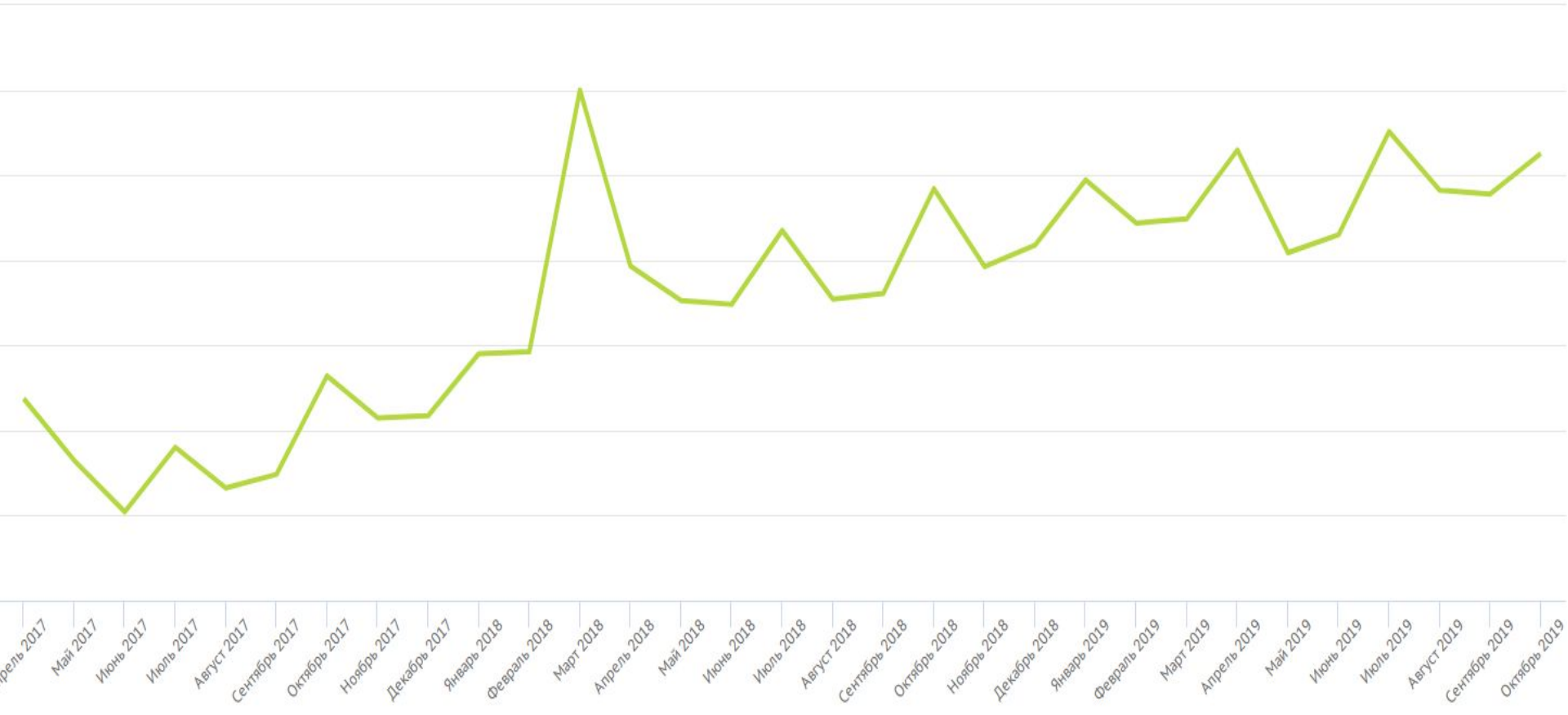
Пользователи выбирают мобильное приложение

- Рынок мобильных приложений растет
- Количество пользователей мобильных приложений растет
- Mobile first

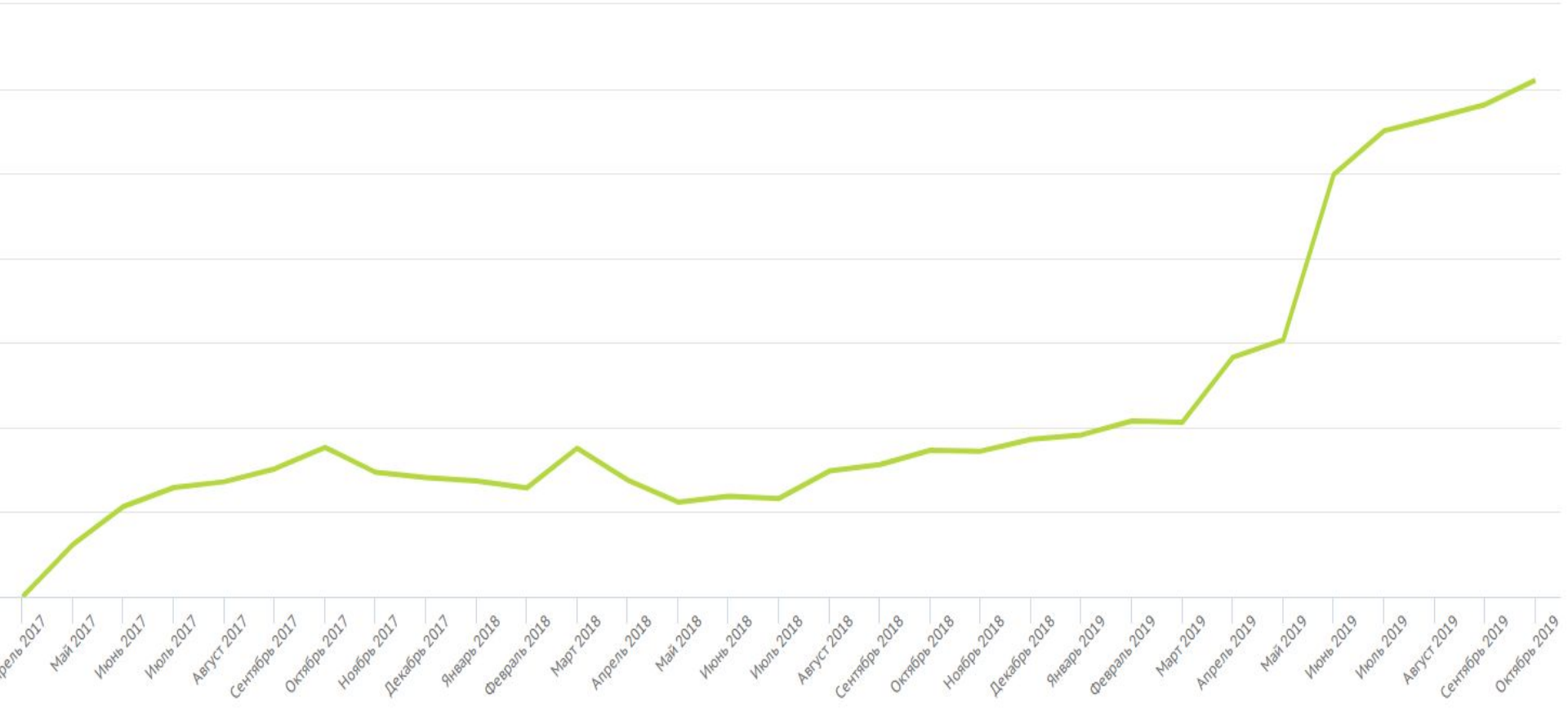


Доля покупок сделанных из приложений и браузеров

Статистика облака СБИС (пользователи браузеров)



Статистика облака СБИС (пользователи приложений)



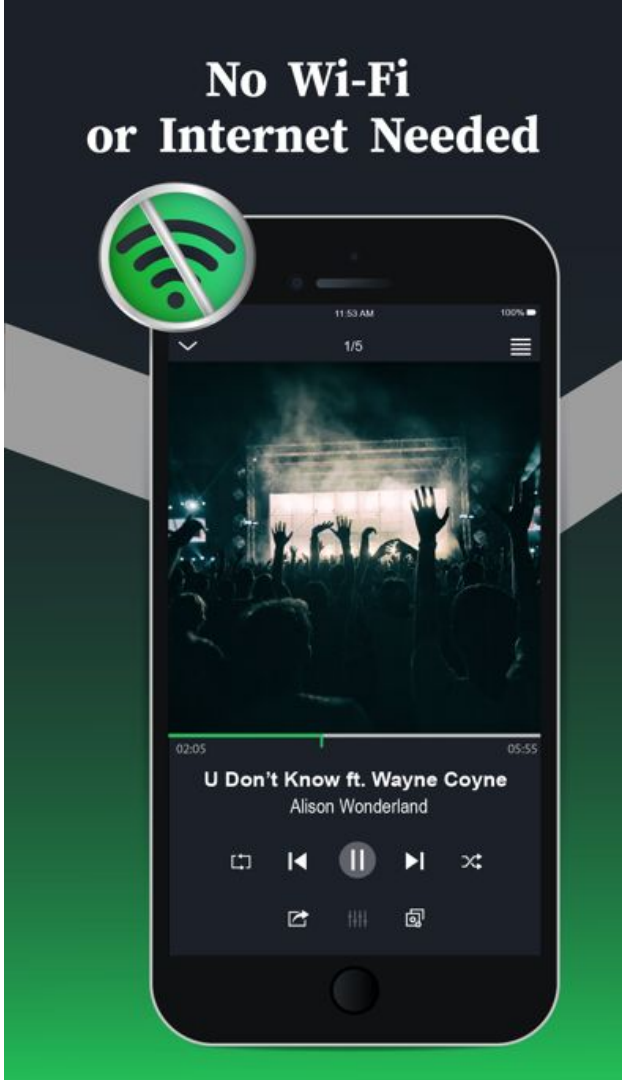
Мобильное приложение

=

Offline приложение со своей бизнес-логикой

≠

"Просто способ организации данных"©



Предложения от гуру мобильной разработки

Набираем команду разработки под Android

Набираем команду разработки под iOS

Берем VIPER

Что получили

Реально разное поведение мобильного приложения под Android и iOS

Разные guidelines - разные интерфейсы

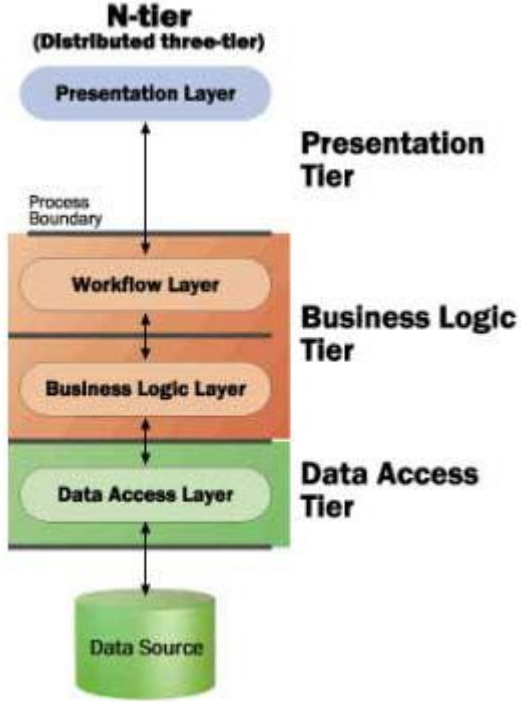
Разные скорости выпуска приложений

Альтернатива - Архитектура

- Мобильное приложение имеет offline логику
- Эта логика посередине от "backend"
- vipEr - эту "E" нельзя сделать как попало

Что такое "backend" и где он должен быть?

- Tier приложения
- Layer приложения
- Tier - вычислительный контейнер
- Layer - логически организованный код
- Правильный layer прозрачно перемещается

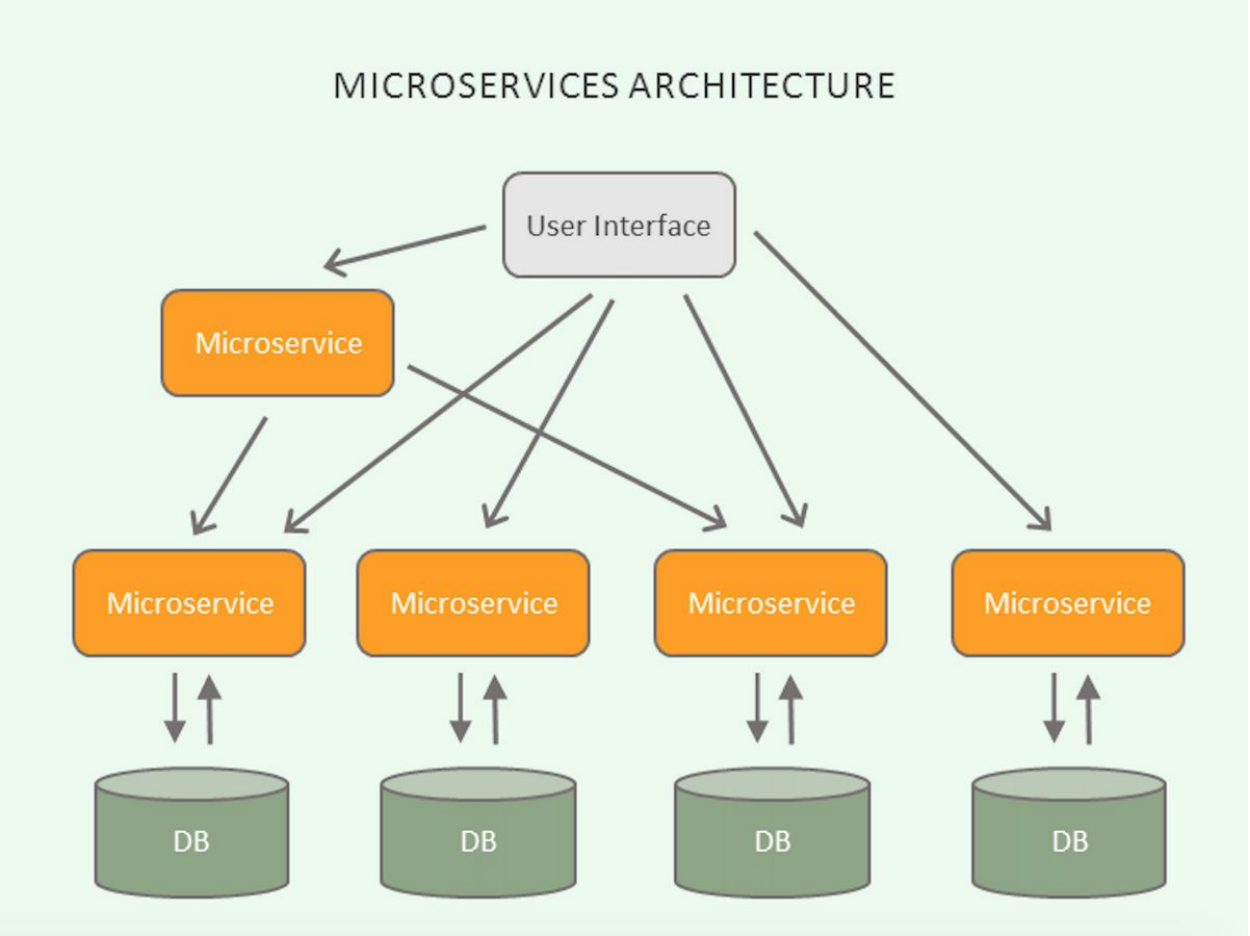


Микросервис - это layer

Backend-программист делает облачный микросервис

Микросервис может потреблять другой микросервис

Клиентский микросервис - умный, с кэшированием



Для разных tier разные имплементации микросервиса

- "Облачная" имплементация - широкий канал -> упор на производительность
- Offline имплементация - слабый канал -> упор на минимизацию трафика

Backend мобильного приложения должен быть в мобильном приложении

Разработчики backend - полноценные участники мобильной разработки

Делаем offline имплементации облачных микросервисов

Offline имплементации пишем на c++, нужна кросскомпиляция

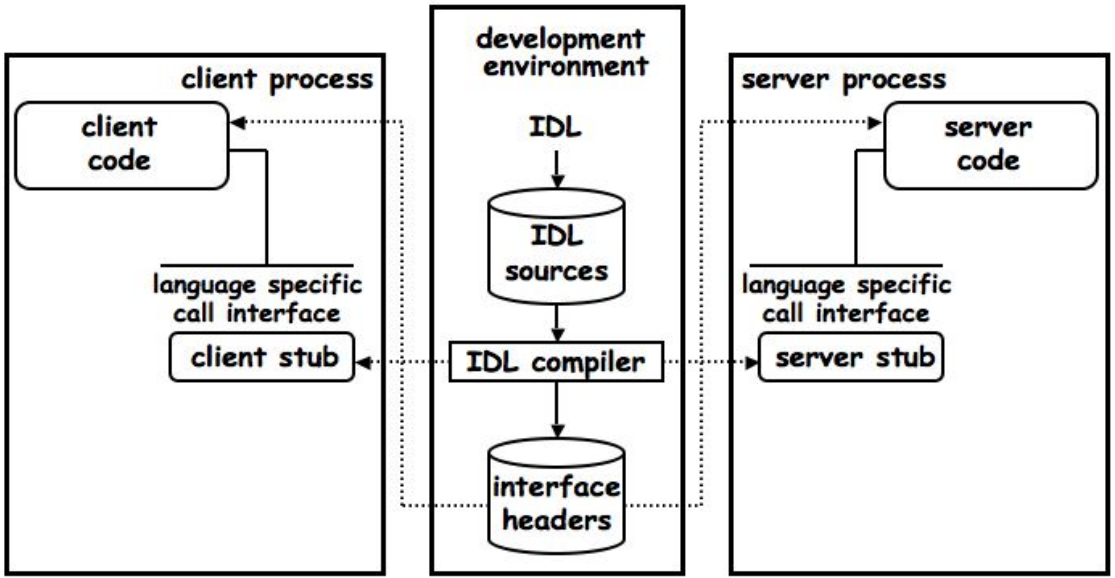
Фаулер одобряет

- Физические границы обеспечены (c++ же)
- Данные децентрализованы (каждому сервису своя БД)
- Автотестирование сервисов налажено (на desktop-консоли прямо)

Изоляция микросервисов

1. Для физической изоляции пишем на c++
2. Фасады микросервисов описываем на IDL
3. Кодогенерация remote stubs, а также мостов Swift/Kotlin для мобильных UI-клиентов

IDL


development environment
IDL
IDL sources
IDL compiler
interface headers
client process
client code
language specific call interface
client stub
server process
server code
language specific call interface
server stub

Децентрализация данных

1. Каждый микросервис имеет отдельную БД
2. В качестве СУБД - SQLite
3. Датамодели из IDL используются не только для remoting, но также для типизированной работы с сущностями БД

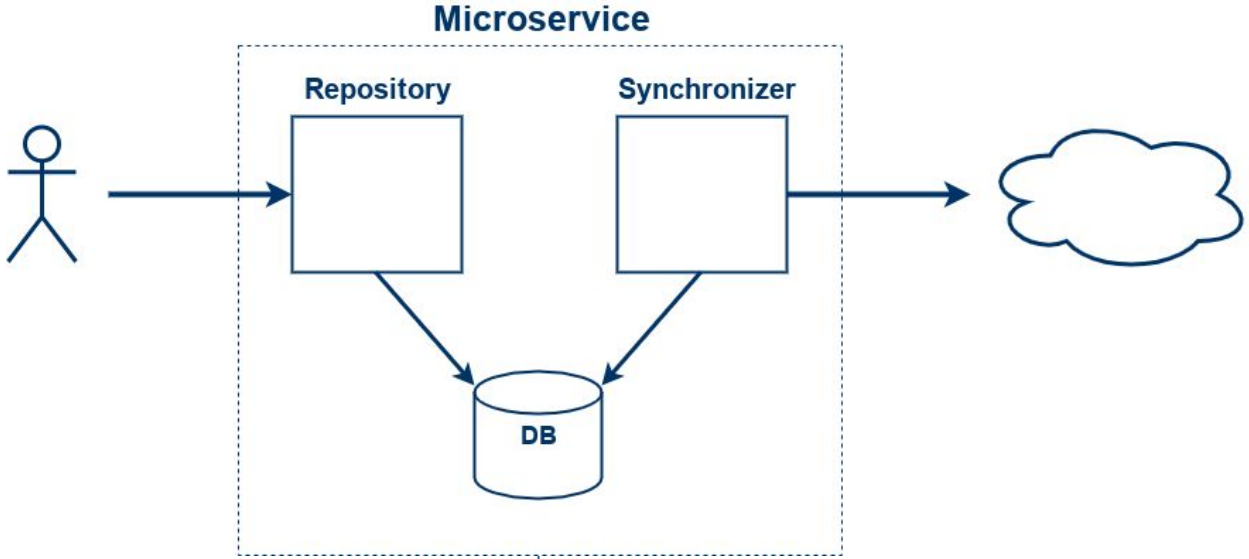
Continuous Integration

1. Каждый микросервис имеет свой набор тестов
2. Test-suite микросервиса собирается под x86_64 архитектуру
3. Тесты запускаются в консоли тестового фреймоворка, сторонние сервисы - mock-реализация

Слои микросервиса

Слой для потребителя

Слой для источника данных



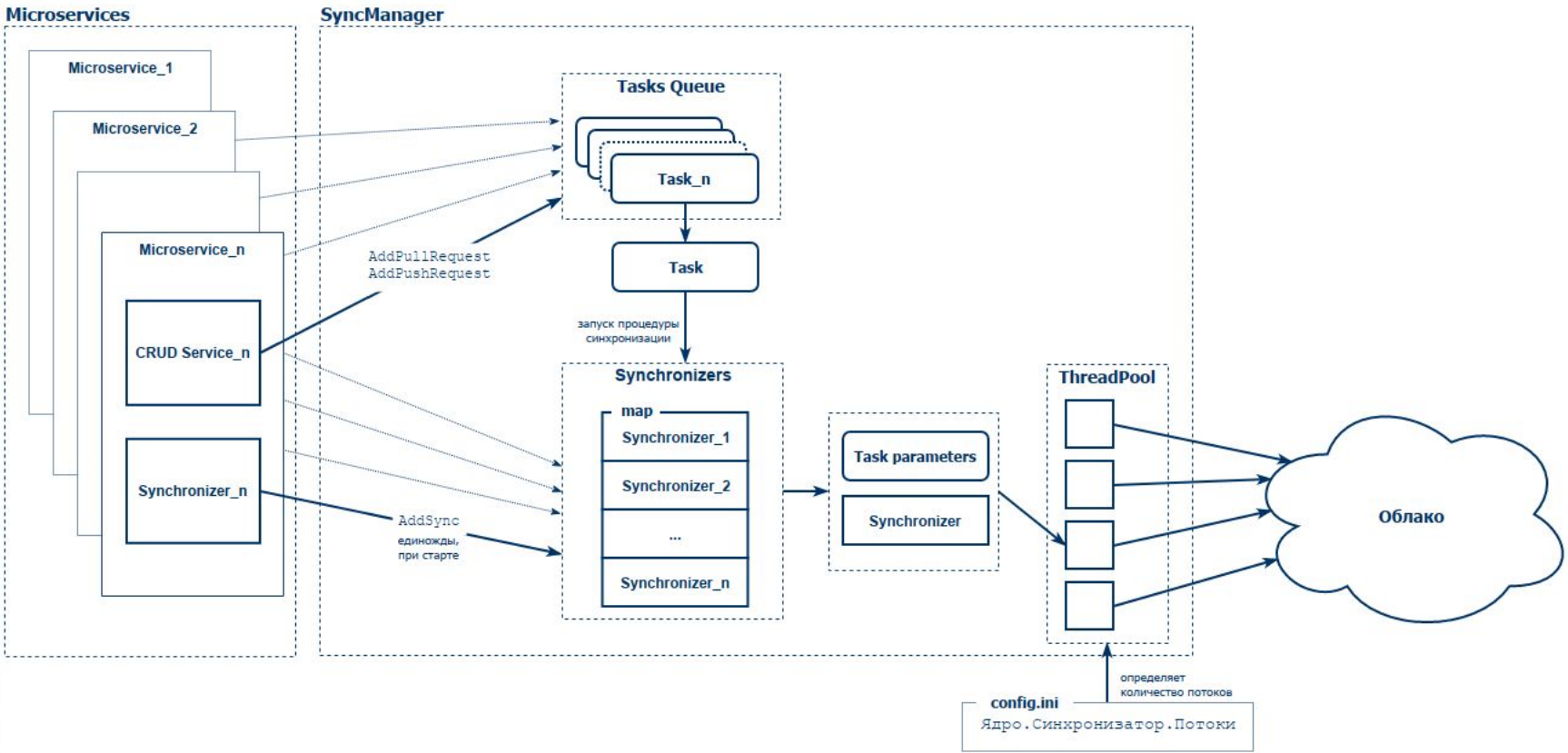
Слой для потребителя

1. Этот слой реализован по design pattern - Repository
2. Потребителем может выступать микросервис и человек через UI
3. Потребитель получает API в стиле CRUD
4. Во всяческих MVC/MVP/MVVM - это M (модель)

Слой для источника данных

1. Этот слой инкапсулирует в себе все взаимодействие с удаленным источником данных
2. Получение данных представляет собой синхронизацию, т. е. микросервис старается получить дельту изменений
3. БД в данном случае выступает как шина для взаимодействия между слоями, но на практике между слоями есть зацепление через API

Чуть больше внимание синхронизатору

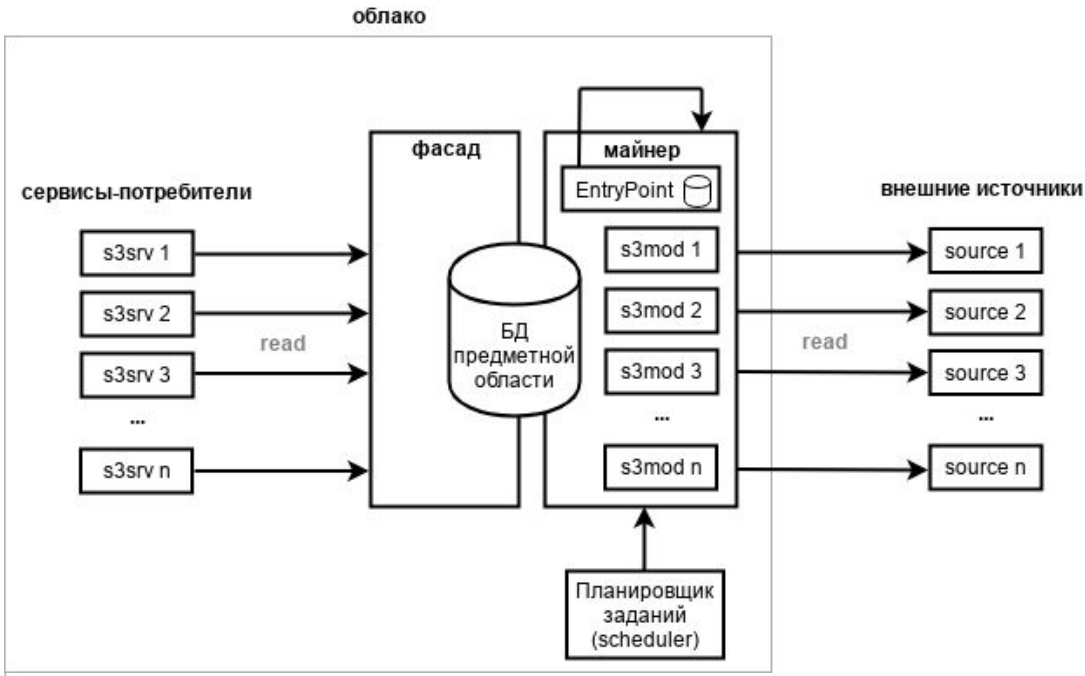


Общий класс задач асинхронного получения данных

1. Offline микросервис внутри мобильного приложения решает задачу частичной репликации данных
2. К подобному классу относятся задачи некоторых сервисов облачного backend
3. Когда нужно получить данные из внешнего, с точки зрения облака, источника
4. Например, организация микросервиса контрагентов

Масштабируем обратно

- Переносим каркасное решение обратно в облако
- Слои микросервиса становятся полноценными сервисами
- Сервис-фасад - это repository
- Сервис-майнер - это synchronizer



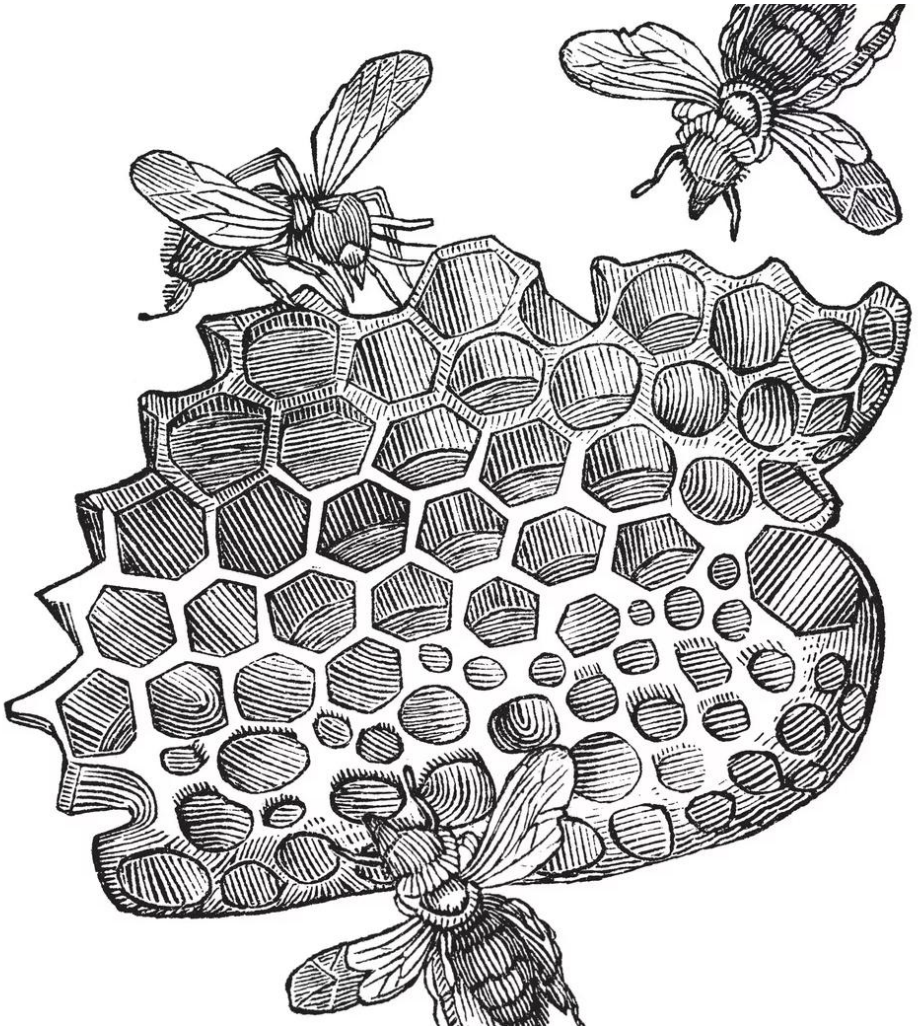
Проблемы подхода

c++ программисты?

Нужно быть большим и сильным

В заключение

- Главное правильно организованный сервис
- Правильно организованному сервису без разницы на каком tier выполняться



Спасибо за внимание.
Вопросы?

Голованов Константин
Руководитель разработки платформы
golovanov.ka@tensor.ru